졸업논문청구논문

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

박 서 진 (□ □ □ Park, Seo Jin)

19039

과학영재학교 경기과학고등학교

2022

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

[논문제출 전 체크리스트]

1. 이 논문은 내가 직접 연구하고 작성한 것이다. ☑
2. 인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다. ☑
3. 인용한 자료의 표현이나 내용을 왜곡하지 않았다. ☑
4. 정확한 출처제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다. ☑
5. 논문 작성 중 도표나 데이터를 조작(위조 혹은 변조)하지 않았다. ☑
6. 다른 친구와 같은 내용의 논문을 제출하지 않았다. ☑

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

Advisor : Teacher Park, Kiehyun

by

19039 **Park, Seo Jin**

Gyeonggi Science High School for the gifted

A thesis submitted to the Gyeonggi Science High School in partial fulfillment of the requirements for the graduation. The study was conducted in accordance with Code of Research Ethics.*

2021. 8. 3.

Approved by
Teacher Park, Kiehyun
[Thesis Advisor]

*Declaration of Ethical Conduct in Research: I, as a graduate student of GSHS, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

박 서 진

위 논문은 과학영재학교 경기과학고등학교 졸업논문으로
졸업논문심사위원회에서 심사 통과하였음.

2021년 8월 3일

심사위원장 김 학 성 (인)

심사위원 이 호 (인)

심사위원 박 기 현 (인)

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

Abstract

Put your abstract here. It is completely consistent with 한글초록.

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

초 록

초록(요약문)은 가장 마지막에 작성한다. 연구한 내용, 즉 본론부터 요약한다. 서론 요약은 하지 않는다. 대개 첫 문장은 연구 주제 (+방법을 핵심적으로 나타낼 수 있는 문구: 실험적으로, 이론적으로, 시뮬레이션을 통해)를 쓴다. 다음으로 연구 방법을 요약한다. 선행 연구들과 구별되는 특징을 중심으로 쓴다. 뚜렷한 특징이 없다면 연구방법은 안써도 상관없다. 다음으로 연구 결과를 쓴다. 연구 결과는 추론을 담지 않고, 객관적으로 서술한다. 마지막으로 결론을 쓴다. 이 연구를 통해 주장하고자 하는 바를 간략히 쓴다. 요약문 전체에서 연구 결과와 결론이 차지하는 비율이 절반이 넘도록 한다. 읽는 이가 요약문으로부터 얻으려는 정보는 연구 결과와 결론이기 때문이다. 연구 결과만 레포트하는 논문인 경우, 결론을 쓰지 않는 경우도 있다.

Contents

List of Tables

List of Figures

I. 서론

I.1 연구의 필요성 및 목적

복잡하게 구성된 지구의 순환 체계에서 Sea Surface Temperature (SST)는 빠뜨릴 수 없는 요소이다. 기후에 밀접하게 영향을 주고받는 SST는 몇몇 해역에서 대기에 강제력을 행사하고, 다른 해역에서는 대기에 영향을 받으며 억지력으로서 작용한다. 계절에 따라 SST와 대기가 미치는 영향의 비중이 달라지는 해역도 존재한다 [1].

SST는 태풍이나 집중호우 등의 위험기상의 발생가능성 또한 SST의 변동성과 연관지어 예측할 수 있는 만큼 SST를 관측하고 그 경향성을 파악하는 것은 지구 환경을 이해하는 데에 굉장히 중요하다 [2].

SST를 관측하는 방법으로는 크게 해양 부이를 이용한 관측과 인공위성 자료를 통한 산출법이 있다. 전자의 경우 구름과 같은 오차 원인을 배제하고 직접적으로 정확한 데이터를 얻을 수 있다는 장점이 있으나, 부이가 위치하는 한 점의 값만을 얻을 수 있기 때문에 폭넓은 지역의 해수면 온도를 알 수 없다는 단점이 있다. 그와는 반대로 인공위성 자료를 통한 산출법은 대기와 다른 여러 요인들로 인한 오차를 계산해야 하나, 위성으로 관측할 수 있는 광범위한 해역의 정보를 알 수 있다는 것이 장점이다.

본 연구에서는 인공위성 자료를 이용하여 한반도 주변 해역의 SST를 산출해 보고자 한다.

I.2 이론적 배경

I.2.1 기상 위성

기상위성이란 지구의 기상현상과 대기를 관측하기 위한 목적의 인공위성들의 분류이며, 우리가 현재 사용하는 기상위성은 궤도에 따라 정지궤도위성과 극궤도위성으로 나뉜다.

정지궤도위성은 적도 상공에 위치해, 약 35,800 km 높이에서 지구와 같은 각속도로 지구 주위를 공전하기 때문에 지상의 관측자가 보았을 때에는 하늘에 고정된 것처럼 느껴지므로 이와 같은 명칭이 붙었다. 정지궤도위성은 지구의 약 $\frac{1}{4}$ 정도 되는 고정된 면적을 관측할 수 있으며 이 때문에 한 지역의 연속적인 기상 상태 변화 등을 관찰하는 데에 있어 유용하다.

극궤도위성은 남극과 북극을 통과하여 지구 주위를 공전하는 위성으로, 고도는 약 800 – 1,500 km 정도이다. 이는 하루에 전체 지구를 약 2회 관측할 수 있으며, 고도가 기상위성에 비해 낮아 세기가 약한 파장도 인식할 수 있으며, 극지의 얼음, 해양, 에너지의 순환 등 다양한 현상을 관측할 수 있다.

I.2.2 NOAA 위성

National Oceanic and Atmospheric Administration (NOAA)에서 진행하는 Polar Operational Environmental Satellite (POES) 프로젝트의 일부로 NOAA 위성을 운용하고 있다. 이 위성은 직하점을 중심으로 55.4° 안쪽의 범위를 주사할 수 있다. 탑재되어 있는 주 관측 센서는 Advanced Very High Resolution Radiometer (AVHRR)와 Television InfraRed Observation Satellite Operational Vertical Sounder (TOVS) 등이 있다. 이 가운데 AVHRR은 5개의 채널을 가졌으며 각각의 파장과 주 용도는 Table 1과 같다.

Table 1. Description for AVHRR channels Channel.

Channel Number	Wavelength (μm)	Typical Use
1	0.58 ~0.68	Daytime cloud and surface mapping
2	0.725 ~1.00	Land-water boundaries
3a	1.58 ~1.64	Snow and ice detection
3b	3.55 ~3.93	Night cloud mapping, Sea surface temperature
4	10.30 ~11.30	Night cloud mapping, Sea surface temperature
5	11.50 ~12.50	Sea surface temperature

I.2.3 Terra/Aqua 위성

1999년 12월 18일 발사되어 2000년 2월 24일 부터 자료를 송신한 Terra (EOS AM-1) 위성은 하루에 한 지점을 2번 관측하는 극궤도위성이다. 지구 환경과 기후의 변화를 관측하는 것이 목표인 이 위성은 Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), Clouds and the Earth's Radiant Energy System (CERES), Multi-angle Imaging SpectroRadiometer (MISR), Moderate-resolution Imaging Spectroradiometer (MODIS), Measurements of Pollution in the Troposphere (MOPITT) 로 총 6 가지의 센서들을 탑재하였다.

Aqua 위성은 2002년 5월 4일 지표면과 대기 중의 물에 관한 연구를 위하여 발사되었으며, Atmospheric Infrared Sounder (AIRS), the Advanced Microwave Sounding Unit (AMSU-A), the Humidity Sounder for Brazil (HSB), the Advanced Microwave Scanning Radiometer for EOS (AMSR-E), the Moderate-Resolution Imaging Spectroradiometer (MODIS), and the Clouds and the Earth's Radiant Energy System (CERES) 로 총 6가지 센서들을 탑재하였으나, 그중 AMSR-E와 HSB가 손상되어 작동을 멈추었고, AMSU-A와 CERES는 일부 고장이 발생하였으나 여전히 작동하고 있다. Terra와 Aqua 위성은 Aura 위성과 함께 Earth Observing System(EOS)의 일부이다.

MODIS는 Terra와 Aqua 위성의 핵심 탑재체이다. 크기 1.0 m × 1.6 m × 1.0 m, 질량 228.7 kg의 MODIS는 위성에 탑재되어 705 km의 고도에서 55°의 시야각, 2330 km의 관측폭으로 하루 한 번 혹은 두 번 같은 지점을 관측한다. 총 36 개인 각 채널의 해상도는 각각 250 m(채널 1 - 2), 500 m(채널 3 - 7), 1 km(채널 8 - 36)이며 그 중 SST 관측에 쓰이는 것은 약 3.7 – 4.1 μm의 대역폭을 가지고 있는 20, 21, 22, 23 번 채널과 10.8 – 12.3 μm의 31, 32 번 채널이다.

I.2.4 인공위성 자료

인공위성 자료는 처리 정도에 따라 레벨 0, 레벨 1A, 레벨 1B, 레벨 2, 레벨 3, 레벨 4 데이터로 나뉜다 [3].

레벨 0 데이터는 우주선에서 지상으로 전송하는 데 쓰이는 통신 정보만을 제거한 상태의 페이로드 데이터를 의미하며, 레벨 1A 데이터는 시간을 참조하여 레벨 0 데이터를 재구성하고 기하적 보정 등 보조 자료를 주석으로 추가한 상태이다. 레벨 1B 데이터는 그것에서 센서의 특성과 복사량에 대한 보정이 이루어진 결과물로, 이 단계부터는 센서 보정이 변경된다면 다른 데이터로 대체되어야만 한다.

레벨 2 데이터는 이들을 이용하여 지구물리학적으로 의미있는 변수들을 도출하여 SST(Sea Surface Temperature), OC(Ocean Color) 등의 그룹으로 분류한 것이고, 레벨 3 데이터는 그러한 데이터를 일정 기간 동안 일정 구역 집계한 기록이다.

마지막으로 레벨 4 데이터는 하위 레벨 데이터에 대한 분석을 말한다.

본 연구에서는 인공위성을 이용한 SST 산출 방식을 채택하여 NOAA 위성의 AVHRR 센서로 관측한 레벨 2 데이터를 레벨 3 데이터로 가공하여 분석하는 것이 목적이다.

I.2.5 SST 산출 알고리즘

인공위성 자료를 통해 SST 데이터를 산출하는 데에는 MCSST(Multi-Channel Sea Surface Temperature)와 CPSST(Cross Product Sea Surface Temperature) 등 여러 기법이 존재한

다. (박경혜, 정종률, 최병호, 김구, 1994) SST 산출에 쓰이는 채널은 22, 23번(단파)와 31, 32번(장파)이며, 각각의 채널에서는 지표면을 흑체로 가정하고 슈테판-볼츠만 법칙을 이용하여 밝기온도를 구한다. McMillin과 Crosby(1984)의 연구 결과에 의하면 수증기흡수계수 ki, kj에 대하여 =kjkj-ki일 때, SST=Tj+(Ti-Tj)의 값을 가진다.

그렇게 도출한 단일채널 SST의 값을 이용하여 아래와 같은 총 세 가지 기법으로 MCSST를 산출한다 [4].

MCSST(3, 4)=T11+1.616(T3.7-T11)+1.07 (dual window) MCSST(4, 5)=T12+3.15(T11-T12)+0.10 (split window) MCSST(3, 4, 5)=T11+0.943(T3.7-T12)+0.61 (triple window)

MCSST를 구하는 식에서는 수증기의 적외선 흡수율이 상수라고 가정하나, 실제로는 온도와 관계 있는 비선형적 함수로서 나타나고, 이에 따라 건조한 극지방이나 고온의 지역에서 산출한 결과와는 오차가 발생하게 된다. 따라서 이를 보완하기 위하여 개발된 비선형 알고리즘이 CPSST이다. (Walton et al, 1998)

II. 연구 방법 및 과정

II.1 데이터 파악 및 수집

해양위성센터에서 제공하는 SST 데이터를 다운받을 수 있는 경로를 확인하였다. 접근할 수 있는 데이터는 2020년 4월 29일 기준으로 Table 과 같다.

2012년부터 2019년까지의 8년 동안 Terra/Aqua 위성이 MODIS를 통해 수집한 자료를 연구에 이용하기로 결정하고, Github에서 다운로드한 웹 크롤링 파일을 이용하여 해양위성센터의 SST 데이터를 크롤링하는 table 과 같이 코드를 작성하였다. 2021년 6월 현재는 천리안위성 2호가 서비스를 시작하면서 사이트가 개편되어 데이터 배포 방식이 바뀌었어 아래의 코드가 실행되지 않을 수도 있다.

온라인 원격수업 환경에서 파일을 다운로드받기 위해 Chrome Remote Desktop을 이용하여 개인 노트북을 Ubuntu 운영체제의 서버 컴퓨터와 연결하여 사용할 수 있도록 하였으며, Ubuntu 프롬프트 명령어를 사용하여 파일 디렉토리를 탐색하는 방법을 학습하였다. 오랜 시간 동안 많은 양의 데이터를 다운받아야 하기 때문에 도중에 프롬프트 창을 닫더라도 계속 다운받을 수 있도록 nohup 명령어를 이용하여 백그라운드로 파일을 실행하였다.

III. Test Section

Title : 21pt, bold face.

III.1 Test Subsection

Title : 16pt, bold face.

III.1.1 Test Subsubsection

Title : 14pt, normal style.

앞서 언급하였듯이 subsubsection, paragraph 와 같은 하위 절은 사용을 자제하는 것이 좋다.

IV. 결론

글이라는 것은 개인의 개성이 담겨 있기 때문에 모든 사람들이 동일한 방식으로 표현하는 것은 아니다. 그러나 고대로부터 개인의 연구 내용을 글로써 타인에게 전달할 때, 효율적인 방법이라고 공감대를 형성하며 다듬어져 온 것이 지금의 논문 형태이다. 그러므로 처음 논문을 작성하는 학생들은 이 문서에서 지시하는 논문 작성 방식을 따르는 것을 권한다. 하지만 여기서는 다양한 논문들에 대해 일일이 사례를 들어 올바른 논문 작성법을 설명하기에는 한계가 있기에 간략하게만 소개를 했다. 여기서 설명되지 않은 부분들은 다른 사람들의 논문을 참고하자. 이미 서론을 작성하면서 많은 선행 연구 논문들을 읽어 봤을 것이다. 그 논문들에서는 데이터를 어떤 방식으로 표현하는지, 서론은 어떤 흐름으로 구성하는지 등을 살펴보자. 논문을 잘 쓰는 비결의 첫 번째는 논문을 많이 읽어 보는 것이다.

+ 첨언을 하자면, 본교의 영어논문작성법 수업에 사용되는 'Science Research Writing for Non-Native Speakers of English' 를 참고하면 많은 도움이 될 것이다.

V. 부록

V.1 Python 코드1

```
'''
#!/usr/bin/env python3
# □*□ coding: utf□8 □*□
Created on Sat Nov  3 20:34:47 2018
@author: guitar79
created by Kevin
#Open hdf file
NameError: name 'SD' is not defined
conda install □c conda□forge pyhdf
'''

from glob import glob
import numpy as np
from datetime import datetime

import os
from pyhdf.SD import SD, SDC

def write_log( log_file , log_str):
    import os
    with open(log_file, 'a') as log_f:
        log_f.write(" {}, □{}₩n".format(os.path.basename(__file__), log_str))
    return print (" {}, □{}₩n".format(os.path.basename(__file__), log_str))

#for checking time
cht_start_time = datetime.now()

#JulianDate_to_date(2018, 131) □□ '20180511'
def JulianDate_to_date(y, jd):
    ############################################################
    #JulianDate_to_date(2018, 131) □□ '20180511'
    #
    month = 1
    while jd □ calendar.monthrange(y, month)[1] > 0 and month <= 12:
        jd = jd □ calendar.monthrange(y, month)[1]
        month += 1
    #return datetime(y, month, jd).strftime('%Y%m%d')
    return datetime(y, month, jd)

def date_to_JulianDate(dt, fmt):
    ############################################################
    #date_to_JulianDate('20180201', '%Y%m%d') □□ 2018032
    #
    dt = datetime.strptime(dt, fmt)
    tt = dt.timetuple()
    return int('%d%03d' % (tt.tm_year, tt.tm_yday))


def fullname_to_datetime_for_DAAC3K(fullname):
    ############################################################
    #for modis hdf file, filename = 'DAAC_MOD04_3K/MOD04_3K.A2014003.0105.006.2015072123557.hdf'
    #
    import calendar
```

```
54      fullname_el = fullname.split("/")
55      filename_el = fullname_el[-1].split(".")
56      y = int(filename_el[1][1:5])
57      jd = int(filename_el[1][5:])
58      month = 1
59      while jd > calendar.monthrange(y, month)[1] and month <= 12:
60          jd = jd - calendar.monthrange(y, month)[1]
61          month += 1
62      #print("filename_el: {}".format(filename_el))
63      #print(y, month, jd, int(filename_el[2][:2]), int(filename_el[2][2:]))
64      return datetime(y, month, jd, int(filename_el[2][:2]), int(filename_el[2][2:]))
65
66
67  def fullname_to_datetime_for_KOSC_MODIS_SST(fullname):
68      ############################################################
69      #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.hdf'
70      #
71      from datetime import datetime
72
73      fullname_info = fullname.split('/')
74       fileinfo  = fullname_info[-1].split('.')
75      filename_dt = datetime(int(fileinfo[-5]), int(fileinfo[-4][:2]), int(fileinfo[-4][2:]), int(fileinfo[-3][:2]),
            int(fileinfo[-3][2:]))
76      return filename_dt
77
78  def fullname_to_datetime_for_KOSC_AVHRR_SST_asc(fullname):
79      ############################################################
80      #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.hdf'
81      #
82      from datetime import datetime
83
84      fullname_info = fullname.split('/')
85       fileinfo  = fullname_info[-1].split('.')
86      filename_dt = datetime(int(fileinfo[0]), int(fileinfo[1][:2]), int(fileinfo[1][2:]), int(fileinfo[2][:2]), int
          (fileinfo[2][2:]))
87      return filename_dt
88
89  def fullname_to_datetime_for_KOSC_MODIS_hdf(fullname):
90      ############################################################
91      #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.hdf'
92      #
93      from datetime import datetime
94
95      fullname_info = fullname.split('/')
96       fileinfo  = fullname_info[-1].split('.')
97      filename_dt = datetime(int(fileinfo[1]), int(fileinfo[2][:2]), int(fileinfo[2][2:]), int(fileinfo[3][:2]), int
          (fileinfo[3][2:]))
98      return filename_dt
99
100
101 def draw_histogram_hdf(hdf_value, longitude, latitude, save_dir_name, fullname, DATAFIELD_NAME):
102     fullname_el = fullname.split("/")
103     import matplotlib.pyplot as plt
104     import numpy as np
105     plt.figure(figsize=(12, 8))
106     plt.title("Histogram-of-{0}:-₩n{1}₩nmean-:-{2:.02f},-max:-{3:.02f},-min:-{4:.02f}₩n₩
107 ----------longigude-:-{5:.02f}~{6:.02f},-latitude:-{7:.02f}~{8:.02f}".format(
        DATAFIELD_NAME, fullname_el[-1],₩
108                                     np.nanmean(hdf_value), np.nanmax(hdf_value), np.nanmin(hdf_value),₩
```

```
109                                         np.nanmin(longitude), np.nanmax(longitude),₩
110                                         np.nanmin(latitude), np.nanmax(latitude)), fontsize=9)
111     plt.hist(hdf_value)
112     plt.grid(True)
113
114     return plt
115
116 def draw_histogram(hdf_value, longitude, latitude, save_dir_name, fullname, DATAFIELD_NAME):
117     fullname_el = fullname.split("/")
118     import matplotlib.pyplot as plt
119     import numpy as np
120     plt.figure(figsize=(12, 8))
121     plt.title("Histogram□of□{0}:□₩n{1}₩nmean□:□{2:.02f},□max:□{3:.02f},□min:□{4:.02f}₩n₩
122 □□□□□□□□□□□□□□□□longigude□:□{5:.02f}~{6:.02f},□latitude:□{7:.02f}~{8:.02f}".format(
        DATAFIELD_NAME, fullname_el[□1],₩
123                                         np.nanmean(hdf_value), np.nanmax(hdf_value), np.nanmin(hdf_value),₩
124                                         np.nanmin(longitude), np.nanmax(longitude),₩
125                                         np.nanmin(latitude), np.nanmax(latitude)), fontsize=9)
126     plt.hist(hdf_value)
127     plt.grid(True)
128     plt.savefig("{0}{1}_{2}_hist.png"₩
129         .format(save_dir_name, fullname_el[□1][:□4], DATAFIELD_NAME))
130     print("{0}{1}_{2}_hist.png□is□created..."₩
131         .format(save_dir_name, fullname_el[□1][:□4], DATAFIELD_NAME))
132     plt.close()
133     return None
134
135
136 def draw_map_MODIS_hdf(hdf_value, longitude, latitude, save_dir_name, fullname, DATAFIELD_NAME, Llon,
      Rlon, Slat, Nlat):
137     fullname_el = fullname.split("/")
138     import numpy as np
139     #if np.isnan(hdf_value).any() :
140     #    print("(np.isnan(hdf_value).any()) is true ...")
141     #else :
142     from mpl_toolkits.basemap import Basemap
143     import matplotlib.pyplot as plt
144
145     plt.figure(figsize=(10, 10))
146
147     # sylender map
148     m = Basemap(projection='cyl', resolution='l', ₩
149                 llcrnrlat = Slat, urcrnrlat = Nlat, ₩
150                 llcrnrlon = Llon, urcrnrlon = Rlon)
151
152     m.drawcoastlines(linewidth=0.25, color='white')
153     m.drawcountries(linewidth=0.25, color='white')
154     m.fillcontinents(color='black', lake_color='black')
155     m.drawmapboundary()
156
157     m.drawparallels(np.arange(□90., 90., 10.), labels=[1, 0, 0, 0], color='white')
158     m.drawmeridians(np.arange(□180., 181., 15.), labels=[0, 0, 0, 1], color='white')
159
160     x, y = m(longitude, latitude) # convert to projection map
161
162     m.pcolormesh(x, y, hdf_value, vmin=0, vmax=40, cmap='coolwarm')
163     m.colorbar(fraction=0.0455, pad=0.044, ticks=(np.arange(□5, 40.1, step=5)))
164
165     plt.title('MODIS□{}'.format(DATAFIELD_NAME), fontsize=20)
```

```
166
167     x1, y1 = m(Llon, Slat□1.5)
168     plt.text(x1, y1, "Maximun□value:□{0:.1f}₩nMean□value:□{1:.1f}₩nMin□value:□{2:.1f}₩n"₩
169             .format(np.nanmax(hdf_value), np.nanmean(hdf_value),
170                     np.nanmin(hdf_value)),
171             horizontalalignment='left',
172             verticalalignment='top',
173             fontsize=9, style='italic', wrap=True)
174
175     x2, y2 = m(Rlon, Slat□1.5)
176     plt.text(x2, y2, "created□by□guitar79@gs.hs.kr₩nAVHRR□SST□procuct□using□KOSC□data₩n{}"₩
177             .format(fullname_el[□1]),
178             horizontalalignment='right',
179             verticalalignment='top',
180             fontsize=10, style='italic', wrap=True)
181
182     return plt
183
184 def draw_map_AVHRR_SST_asc(df_AVHRR_sst, save_dir_name, fullname, DATAFIELD_NAME, Llon, Rlon, Slat,
        Nlat):
185     fullname_el = fullname.split("/")
186     import numpy as np
187     from mpl_toolkits.basemap import Basemap
188     import matplotlib.pyplot as plt
189
190     width = []
191     for i in range(len(df_AVHRR_sst)□1):
192         #print("index : {}".format(df_AVHRR_sst['latitude'].iloc[i]))
193         if abs(df_AVHRR_sst['latitude'].iloc[i] □ df_AVHRR_sst['latitude'].iloc[i+1]) > 0.001 :
194             width.append(i)
195             #print("index : {}".format(i))
196         if i == 10000 : break
197     longitude = df_AVHRR_sst['longitude'].to_numpy()
198     longitude = np.array(longitude, dtype=np.float32)
199     longitude = longitude.reshape((longitude.shape[0]//(width[0]+1)), width[0]+1)
200     latitude = df_AVHRR_sst['latitude'].to_numpy()
201     latitude = np.array(latitude, dtype=np.float32)
202     latitude = latitude.reshape((latitude.shape[0]//(width[0]+1)), width[0]+1)
203     print("latitude.shape:□{}".format(latitude.shape))
204     print("type(latitude)□:□{}".format(type(latitude)))
205     print("latitude:□{}".format(latitude))
206     print("np.nanmax(latitude):□{}".format(np.nanmax(latitude)))
207     print("np.nanmin(latitude):□{}".format(np.nanmin(latitude)))
208
209     sst = df_AVHRR_sst['sst'].to_numpy()
210     sst = np.array(sst, dtype=np.float32)
211     sst = sst.reshape((sst.shape[0]//(width[0]+1)), width[0]+1)
212
213     #Plot data on the map
214     print("="*80)
215     print("Plotting□data□on□the□map")
216
217     plt.figure(figsize=(10, 10))
218
219     # sylender map
220     m = Basemap(projection='cyl', resolution='l', ₩
221                 llcrnrlat = Slat, urcrnrlat = Nlat, ₩
222                 llcrnrlon = Llon, urcrnrlon = Rlon)
223
```

```
224     m.drawcoastlines(linewidth=0.25, color='white')
225     m.drawcountries(linewidth=0.25, color='white')
226     m. fillcontinents (color='black', lake_color='black')
227     m.drawmapboundary()
228
229     m.drawparallels(np.arange(□90., 90., 10.), labels=[1, 0, 0, 0], color='white')
230     m.drawmeridians(np.arange(□180., 181., 15.), labels=[0, 0, 0, 1], color='white')
231
232     x, y = m(longitude, latitude) # convert to projection map
233
234     m.pcolormesh(x, y, sst, vmin=0, vmax=40, cmap='coolwarm')
235     m.colorbar(fraction=0.0455, pad=0.044, ticks=(np.arange(□5, 40.1, step=5)))
236
237     plt. title ('AVHRR□SST', fontsize=20)
238
239     x1, y1 = m(Llon, Slat□1.5)
240     plt.text(x1, y1, "Maximun□value:□{0:.1f}₩nMean□value:□{1:.1f}₩nMin□value:□{2:.1f}₩n"₩
241             .format(np.nanmax(df_AVHRR_sst["sst"]), np.nanmean(df_AVHRR_sst["sst"]),
242                     np.nanmin(df_AVHRR_sst["sst"])),
243             horizontalalignment='left',
244             verticalalignment='top',
245             fontsize=9, style=' italic ', wrap=True)
246
247     x2, y2 = m(Rlon, Slat□1.5)
248     plt.text(x2, y2, "created□by□guitar79@gs.hs.kr₩nAVHRR□SST□procuct□using□KOSC□data₩n{}"₩
249              .format(fullname_el[□1]),
250             horizontalalignment='right',
251             verticalalignment='top',
252             fontsize=10, style=' italic ', wrap=True)
253
254     return plt
255
256
257 def draw_histogram_AVHRR_SST_asc(df_AVHRR_sst, save_dir_name, fullname, DATAFIELD_NAME):
258     import matplotlib.pyplot as plt
259     import numpy as np
260     plt.figure(figsize=(12, 8))
261     plt. title ("Histogram□of□{0}:□₩n{1}₩nmean□:□{2:.02f},□max:□{3:.02f},□min:□{4:.02f}₩n₩
262 □□□□□□□□□□□□□□□longigude□:□{5:.02f}~{6:.02f},□latitude:□{7:.02f}~{8:.02f}".format(
        DATAFIELD_NAME, fullname,₩
263                                     np.nanmean(df_AVHRR_sst["sst"]), np.nanmax(df_AVHRR_sst["sst"]), np.
        nanmin(df_AVHRR_sst["sst"]),₩
264                                     np.nanmin(df_AVHRR_sst["longitude"]), np.nanmax(df_AVHRR_sst["
        longitude"]),₩
265                                     np.nanmin(df_AVHRR_sst["latitude"]), np.nanmax(df_AVHRR_sst["latitude"
        ])), fontsize=9)
266     plt.hist(df_AVHRR_sst["sst"])
267     plt.grid(True)
268
269     return plt
270
271 def draw_histogram_AVHRR_SST_asc1(df_AVHRR_sst, save_dir_name, fullname, DATAFIELD_NAME):
272     fullname_el = fullname.split("/")
273     import matplotlib.pyplot as plt
274     import numpy as np
275     plt.figure(figsize=(12, 8))
276     plt. title ("Histogram□of□{0}:□₩n{1}₩nmean□:□{2:.02f},□max:□{3:.02f},□min:□{4:.02f}₩n₩
277 □□□□□□□□□□□□□□□longigude□:□{5:.02f}~{6:.02f},□latitude:□{7:.02f}~{8:.02f}".format(
        DATAFIELD_NAME, fullname,₩
```

```
                                        np.nanmean(df_AVHRR_sst["sst"]), np.nanmax(df_AVHRR_sst["sst"]), np.
    nanmin(df_AVHRR_sst["sst"]),₩
                                        np.nanmin(df_AVHRR_sst["longitude"]), np.nanmax(df_AVHRR_sst["
    longitude"]),₩
                                        np.nanmin(df_AVHRR_sst["latitude"]), np.nanmax(df_AVHRR_sst["latitude"
    ])), fontsize=9)
    plt.hist(df_AVHRR_sst["sst"])
    plt.grid(True)

    #plt.savefig("{}_{}_hist.png".format(fullname[:-4], DATAFIELD_NAME))
    #print("{}_{}_hist.png is created ...". format(fullname[:-4], DATAFIELD_NAME))
    plt.savefig("{0}{1}_{2}_hist.png"₩
        .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
    print("{0}{1}_{2}_hist.png is created..."₩
        .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
    plt.close()
    return None

def npy_filename_to_fileinfo(fullname):
    ##################################################################
    # for modis hdf file, filename = 'DAAC_MOD04_3K/daily/sst_20110901_20110902_110_150_10_60_0.05
    _alldata.npy'
    #
    fileinfo = fullname.split('_')
    start_date = fileinfo[-8]
    end_date = fileinfo[-7]
    Llon = fileinfo[-6]
    Rlon = fileinfo[-5]
    Slat = fileinfo[-4]
    Nlat = fileinfo[-3]
    resolution = fileinfo[-2]
    return start_date, end_date, Llon, Rlon, Slat, Nlat, resolution

def getFullnameListOfallFiles(dirName):
    ##############################################3
    import os
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = sorted(os.listdir(dirName))
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + getFullnameListOfallFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles


def calculate_mean_using_result_array(result_array):
    mean_array = result_array.copy()
    cnt_array = result_array.copy()
    for i in range(np.shape(result_array)[0]):
        for j in range(np.shape(result_array)[1]):

            if len(result_array[i][j])>0: mean_array[i][j] = np.mean(result_array[i][j])
```

```
333                 else : mean_array[i][j] = np.nan
334                 cnt_array[i][j] = len(result_array[i][j])
335
336     mean_array = np.array(mean_array)
337     cnt_array = np.array(cnt_array)
338     return mean_array, cnt_array
339
340 def make_grid_array(Llon, Rlon, Slat, Nlat, resolution) :
341     ############################################################
342     # Llon, Rlon = 90, 150
343     # Slat, Nlat = 10, 60
344     # resolution = 0.025
345     #
346
347     import numpy as np
348
349     ni = np.int((Rlon-Llon)/resolution+1.00)
350     nj = np.int((Nlat-Slat)/resolution+1.00)
351     array_data = []
352     for i in range(ni):
353         line_data = []
354         for j in range(nj):
355             line_data.append([])
356         array_data.append(line_data)
357
358     return array_data
359
360
361 def make_grid_array1(Llon, Rlon, Slat, Nlat, resolution) :
362     ############################################################
363     # Llon, Rlon = 90, 150
364     # Slat, Nlat = 10, 60
365     # resolution = 0.025
366     #
367
368     import numpy as np
369
370     ni = np.int((Rlon-Llon)/resolution+1.00)
371     nj = np.int((Nlat-Slat)/resolution+1.00)
372     array_lon = []
373     array_lat = []
374     array_data = []
375     for i in range(ni):
376         line_lon = []
377         line_lat = []
378         line_data = []
379         for j in range(nj):
380             line_lon.append(Llon+resolution*i)
381             line_lat.append(Nlat-resolution*j)
382             line_data.append([])
383         array_lon.append(line_lon)
384         array_lat.append(line_lat)
385         array_data.append(line_data)
386     array_lon = np.array(array_lon)
387     array_lat = np.array(array_lat)
388
389     return array_lon, array_lat, array_data
390
391
```

```
392
393 def read_MODIS_hdf_to_ndarray(fullname, DATAFIELD_NAME):
394     ############################################################
395     #
396     #
397     import numpy as np
398     from pyhdf.SD import SD, SDC
399     hdf = SD(fullname, SDC.READ)
400
401     # Read AOD dataset.
402     if DATAFIELD_NAME.upper() in hdf.datasets() :
403         DATAFIELD_NAME = DATAFIELD_NAME.upper()
404
405     if DATAFIELD_NAME in hdf.datasets() :
406         hdf_raw = hdf.select(DATAFIELD_NAME)
407         print("found□data□set□of□{}:□{}".format(DATAFIELD_NAME, hdf_raw))
408
409     else :
410         print("There□is□no□data□set□of□{}:□{}".format(DATAFIELD_NAME, hdf_raw))
411         hdf_raw = np.arange(0)
412
413     # Read geolocation dataset.
414     if 'Latitude' in hdf.datasets() and 'Longitude' in hdf.datasets():
415         lat = hdf.select('Latitude')
416         latitude = lat[:,:]
417         lon = hdf.select('Longitude')
418         longitude = lon[:,:]
419
420     elif 'Latitude'.lower() in hdf.datasets() and 'Longitude'.lower() in hdf.datasets():
421         lat = hdf.select('Latitude'.lower())
422         latitude = lat[:,:]
423         lon = hdf.select('Longitude'.lower())
424         longitude = lon[:,:]
425     else :
426         latitude, longitude ₩
427             = np.arange(0), np.arange(0)
428
429     if 'cntl_pt_cols' in hdf.datasets() and 'cntl_pt_rows' in hdf.datasets():
430         cntl_pt_cols = hdf.select('cntl_pt_cols')
431         cntl_pt_cols = cntl_pt_cols[:]
432         cntl_pt_rows = hdf.select('cntl_pt_rows')
433         cntl_pt_rows = cntl_pt_rows[:]
434     else :
435         cntl_pt_cols, cntl_pt_rows = np.arange(0), np.arange(0)
436
437     return hdf_raw, latitude, longitude, cntl_pt_cols, cntl_pt_rows
438
439
440
441 def read_MODIS_hdf_and_make_statistics_array(dir_name, DATAFIELD_NAME, proc_date,
442                                              resolution, Llon, Rlon, Slat, Nlat):
443
444     proc_start_date = proc_date[0]
445     proc_end_date = proc_date[1]
446     thread_number = proc_date[2]
447     processing_log = '#This□file□is□created□using□python□₩n' ₩
448                      '#https://github.com/guitar79/KOSC_MODIS_SST_Python□₩n' ₩
449                      + '#start□date□=□' + str(proc_date[0]) +'₩n'₩
450                      + '#end□date□=□' + str(proc_date[1]) +'₩n'
```

```
451
452     #convert start_date  and end_date to date type
453     start_date  = datetime(int(proc_start_date[:4]),
454                            int(proc_start_date[4:6]),
455                            int(proc_start_date[6:8]))
456     end_date = datetime(int(proc_end_date[:4]),
457                         int(proc_end_date[4:6]),
458                         int(proc_end_date[6:8]))
459
460     processing_log += '#Llon□=' + str(Llon) + '₩n' ₩
461     + '#Rlon□=' + str(Rlon) + '₩n' ₩
462     + '#Slat□=' + str(Slat)  + '₩n' ₩
463     + '#Nlat□=' + str(Nlat) + '₩n' ₩
464     + '#resolution□=' + str(resolution)  + '₩n'
465
466     print("{0}□{1}□Start□making□grid□arrays...₩n".format(proc_start_date, proc_end_date))
467     ni  = np.int((Rlon□Llon)/resolution+1.00)
468     nj  = np.int((Nlat□Slat)/resolution+1.00)
469     array_lon = []
470     array_lat  = []
471     array_data = []
472     for  i  in  range(ni):
473          line_lon  = []
474          line_lat  = []
475          line_data  = []
476          for  j  in  range(nj):
477               line_lon.append(Llon+resolution*i)
478               line_lat .append(Nlat□resolution*j)
479               line_data.append([])
480          array_lon.append(line_lon)
481          array_lat.append(line_lat)
482          array_data.append(line_data)
483     array_lat  = np.array(array_lat)
484     array_lon = np.array(array_lon)
485     print('Grid□arrays□are□created.....₩n')
486
487     total_data_cnt  = 0
488     file_no  = 0
489     processing_log += '#processing□file□list₩n'
490     processing_log += '#No,□data_count,□filename□₩n'
491
492     result_array  = np.zeros((1,  1,  1))
493     fullnames = sorted(glob(os.path.join(dir_name, '*.hdf')))
494     if  not fullnames  :
495          for  fullname in  fullnames:
496               result_array  = array_data
497               file_date  = fullname_to_datetime_for_MODIS_3K(fullname)
498               #print(' fileinfo ',  file_date )
499
500               if  file_date  >= start_date  ₩
501                    and  file_date  < end_date :
502
503                    try:
504                         print('reading□file□{0}₩n'.format(fullname))
505                         hdf = SD(fullname, SDC.READ)
506                         # Read AOD dataset.
507                         hdf_raw = hdf.select(DATAFIELD_NAME)
508                         hdf_data = hdf_raw[:,:]
509                         scale_factor  = hdf_raw.attributes()['scale_factor']
```

```
                        offset  = hdf_raw.attributes()['add_offset']
                        hdf_value = hdf_data * scale_factor + offset
                        hdf_value[hdf_value < 0] = np.nan
                        hdf_value = np.asarray(hdf_value)

                        # Read geolocation dataset.
                        lat  = hdf.select('Latitude')
                        latitude  = lat[:,:]
                        lon = hdf.select('Longitude')
                        longitude = lon[:,:]
                    except Exception as err :
                        print("Something got wrecked : {}".format(err))
                        continue

                    if np.shape(longitude) != np.shape(latitude) or np.shape(latitude) != np.shape(hdf_value) :
                        print('data shape is different!! \n')
                        print('='*80)
                    else :
                        lon_cood = np.array(((longitude Llon)/resolution*100//100), dtype=np.uint16)
                        lat_cood = np.array(((Nlat latitude)/resolution*100//100), dtype=np.uint16)
                        data_cnt = 0
                        for i in range(np.shape(lon_cood)[0]) :
                            for j in range(np.shape(lon_cood)[1]) :
                                if int(lon_cood[i][j]) < np.shape(array_lon)[0] \
                                    and int(lat_cood[i][j]) < np.shape(array_lon)[1] \
                                    and not np.isnan(hdf_value[i][j]) :
                                    data_cnt += 1 #for debug
                                    result_array[int(lon_cood[i][j])][int(lat_cood[i][j])].append(hdf_value[i][j])
                        file_no += 1
                        total_data_cnt += data_cnt
                    processing_log += str(file_no) + ',' + str(data_cnt) +',' + str(fullname) + '\n'
                    print(thread_number, proc_date[0], 'number of files: ',
                          file_no, ' tatal data cnt :', data_cnt)
            processing_log += '#total data number=' + str(total_data_cnt) + '\n'

        else :
            print("No file exist...")

        return result_array, processing_log



def read_MODIS_SST_hdf_and_array_by_date(save_dir_name, dir_name, proc_date,
                                         resolution, Llon, Rlon, Slat, Nlat):
    add_log = True
    if add_log == True :
        log_file  = 'read_MODIS_AOD_hdf_and_array_by_date.log'
        err_log_file  = 'read_MODIS_AOD_hdf_and_array_by_date_err.log'

    proc_start_date = proc_date[0]
    proc_end_date = proc_date[1]
    thread_number = proc_date[2]
    processing_log = '#This file is created using python \n' \
                     '#https://github.com/guitar79/MODIS_AOD \n' \
                     + '#start date =' + str(proc_date[0]) +'\n'\
                     + '#end date =' + str(proc_date[1]) +'\n'
    #variables for downloading
    start_date  = datetime(int(proc_start_date[:4]),
```

```
                                int(proc_start_date[4:6]),
                                int(proc_start_date[6:8])) #convert startdate to date type
    end_date = datetime(int(proc_end_date[:4]),
                                int(proc_end_date[4:6]),
                                int(proc_end_date[6:8])) #convert startdate to date type

    print('checking... □{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy₩n'₩
            .format(save_dir_name, proc_start_date, proc_end_date,
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)))
    if os.path.exists('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy'₩
            .format(save_dir_name, proc_start_date, proc_end_date,
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))) ₩
        and os.path.exists('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_info.txt'₩
            .format(save_dir_name, proc_start_date, proc_end_date,
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))):

        print('='*80)
        write_log( log_file , '{8}□:::□{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}□files□are□already□exist'₩
                    .format(save_dir_name, proc_start_date, proc_end_date,
                    str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution), datetime.now()))
        return 0

    else :
        processing_log += '#Llon□=' + str(Llon) + '₩n' ₩
        + '#Rlon□=' + str(Rlon) + '₩n' ₩
        + '#Slat□=' + str(Slat) + '₩n' ₩
        + '#Nlat□=' + str(Nlat) + '₩n' ₩
        + '#resolution□=' + str(resolution) + '₩n'

        print('{0}□{1}□Start□making□grid□arrays...₩n'₩
                .format(proc_start_date, proc_end_date))
        ni = np.int((Rlon□Llon)/resolution+1.00)
        nj = np.int((Nlat□Slat)/resolution+1.00)
        array_lon = []
        array_lat = []
        array_data = []
        for i in range(ni):
            line_lon = []
            line_lat = []
            line_data = []
            for j in range(nj):
                line_lon.append(Llon+resolution*i)
                line_lat.append(Nlat□resolution*j)
                line_data.append([])
            array_lon.append(line_lon)
            array_lat.append(line_lat)
            array_data.append(line_data)
        array_lat = np.array(array_lat)
        array_lon = np.array(array_lon)
        print('grid□arrays□are□created.....₩n')

        total_data_cnt = 0
        file_no=0
        processing_log += '#processing□file□list₩n'
        processing_log += '#No,□data_count,□filename□₩n'

        result_array = np.zeros((1, 1, 1))
        for fullname in sorted(glob(os.path.join(dir_name, '*.hdf'))):
```

```
628                 result_array  = array_data
629                 file_date  = fullname_to_datetime_for_MODIS_3K(fullname)
630                 #print(' fileinfo ',  file_date )
631
632                 if  file_date  >= start_date ₩
633                     and file_date  < end_date :
634
635                     try:
636                         print('reading□file□{0}₩n'.format(fullname))
637                         hdf = SD(fullname, SDC.READ)
638                         # Read AOD dataset.
639                         DATAFIELD_NAME = 'Optical_Depth_Land_And_Ocean'
640                         hdf_raw = hdf.select(DATAFIELD_NAME)
641                         hdf_data = hdf_raw[:,:]
642                         scale_factor  = hdf_raw.attributes()['scale_factor']
643                         offset  = hdf_raw.attributes()['add_offset']
644                         hdf_value = hdf_data * scale_factor + offset
645                         hdf_value[hdf_value < 0] = np.nan
646                         hdf_value = np.asarray(hdf_value)
647
648                         # Read geolocation dataset.
649                         lat  = hdf.select('Latitude')
650                         latitude  = lat[:,:]
651                         lon = hdf.select('Longitude')
652                         longitude  = lon[:,:]
653                     except Exception as err :
654                         print("Something□got□wrecked□₩n")
655                         write_log(err_log_file , '{2}□:::□{0}□with□{1}'₩
656                           .format(err,  fullname,  datetime.now()))
657                         continue
658
659                     if  np.shape(longitude) != np.shape(latitude)  or np.shape(latitude)  != np.shape(hdf_value) :
660                         print('data□shape□is□different!!□₩n')
661                         print('='*80)
662                     else  :
663                         lon_cood = np.array(((longitude□Llon)/resolution*100//100), dtype=np.uint16)
664                         lat_cood = np.array(((Nlat□latitude)/resolution*100//100), dtype=np.uint16)
665                         data_cnt = 0
666                         for  i  in  range(np.shape(lon_cood)[0]) :
667                             for  j  in  range(np.shape(lon_cood)[1]) :
668                                 if  int(lon_cood[i][j])  < np.shape(array_lon)[0] ₩
669                                     and int(lat_cood[i][j])  < np.shape(array_lon)[1] ₩
670                                     and not np.isnan(hdf_value[i][j])  :
671                                     data_cnt += 1 #for debug
672                                     result_array[int(lon_cood[i][j])][int(lat_cood[i][j])].append(hdf_value[i][j])
673                         file_no  += 1
674                         total_data_cnt  += data_cnt
675                     processing_log += str(file_no) + ',' + str(data_cnt) +',' + str(fullname) + '₩n'
676                     print(thread_number,  proc_date[0], 'number□of□files:□',
677                           file_no,  'tatal□data□cnt□:', data_cnt)
678             processing_log += '#total□data□number□=' + str(total_data_cnt) + '₩n'
679
680             np.save('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy'₩
681                     .format(save_dir_name, proc_start_date, proc_end_date,
682                     str(Llon),  str(Rlon),  str(Slat),  str(Nlat),  str(resolution)),  result_array)
683
684             with open('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_info.txt'₩
685                       .format(save_dir_name, proc_start_date, proc_end_date,
686                       str(Llon),  str(Rlon),  str(Slat),  str(Nlat),  str(resolution)),  'w') as  f:
```

```
687                 f.write(processing_log)
688             print('#'*60)
689             write_log( log_file , '{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}□files□are□is□created.'₩
690                     .format(save_dir_name, proc_start_date, proc_end_date,
691                         str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)))
692
693         return 0 # Return a dummy value
694         # Putting large values in Queue was slow than expected(~10min)
695         #return result_array , processing_log
```

```
1
2
```

References

[1] Wu, R., & Kirtman, B. P. (2007). Regimes of seasonal air–sea interaction and implications for performance of forced simulations. Climate dynamics, 29(4), 393–410.

[2] 정은실 (2019). 한반도에서 위험기상 발생 시 나타나는 해수면온도 변동의 특성. *한국지구과학회지*, 40(3), 240–258.

[3] Product definitions. `https://oceancolor.gsfc.nasa.gov/products/`. Accessed: 2021-06-30.

[4] Walton, C. C. (1988). Nonlinear multichannel algorithms for estimating sea surface temperature with avhrr satellite data. Journal of Applied Meteorology and Climatology, 27(2), 115–124.